Sentinel Legacy Codeless Connector Platform

PROOF OF CONCEPT - STEP BY STEP GUIDE

THIS GUIDE SHOWS HOW TO CREATE A POC OF A SENTINEL LEGACY CODELESS CONNECTOR - USEFUL TO DEMONSTRATE IN YOUR PERSONAL ENVIRONMENT

Contents

INTRODUCTION	3
ARCHITECTURE	5
CONNECTOR	7
REFERENCE	11

INTRODUCTION

It is possible to collect data in Sentinel using agent based solutions (Microsoft Monitoring Agent, Azure Monitor Agent) or agentless (Diagnostic Settings, Sentinel Rest APIs – and therefore Function Apps, Logic App, any script that allows API calls).

This is independent against the Sentinel Data Connector concept. The Data Connector is simply a tool that makes it easier for us to collect data – sometimes giving step-by-step guides, other times completely hiding the logic for collecting data by showing just a single 'connect' button.

Example

Entra ID Connector: you select the data of interest and with a simple 'Active' button you're done. After less than 20 minutes, the data is available below in Sentinel.

(Spoiler: Entra ID Connector is based on Diagnostic Settings. After activating it you will find a new rule in Entra > Diagnostic Settings > AzureSentinel_nameOfLAW.

Azure Activity Connector - is based on assigning a DeployIfNotExists policy.

Google Cloud Platform IAM (using Azure Functions) Connector - as the name suggests, with a simple click an Azure Function is deployed.

Data connectors are available in the Content Hub, Github or you can create your own custom connector – documentation here > Resources for creating Microsoft Sentinel custom connectors | Microsoft Learn

Method description	Capability	Serverless	Complexity
Codeless Connector Platform (CCP)	Supports all capabilities available with	Yes	Low; simple,
Best for less technical audiences to create	the code.		codeless
SaaS connectors using a configuration file			development
instead of advanced development.			
Log Analytics Agent	File collection only	No	Low
Best for collecting files from on-premises			
and laaS sources			
<u>Logstash</u>	Available plugins, plus custom plugin,	No; requires a VM or	Low; supports many
Best for on-premises and laaS sources, any	capabilities provide significant	VM cluster to run	scenarios with
source for which a plugin is available, and	flexibility.		plugins
organizations already familiar with			
Logstash			
Logic Apps	Codeless programming allows for	Yes	Low; simple,
High cost; avoid for high-volume data	limited flexibility, without support for		codeless
Best for low-volume cloud sources	implementing algorithms.		development
	If no available action already supports		
	your requirements, creating a custom		
	action may add complexity.		
<u>PowerShell</u>	Direct support for file collection.	No	Low
Best for prototyping and periodic file			
uploads	PowerShell can be used to collect		
	more sources, but will require coding		
	and configuring the script as a service.		
Log Analytics API	Supports all capabilities available with	Depends on the	High
Best for ISVs implementing integration,	the code.	implementation	
and for unique collection requirements			
Azure Functions	Supports all capabilities available with	Yes	High; requires
Best for high-volume cloud sources, and	the code.		programming
for unique collection requirements			knowledge

But what is the Codeless Connector Platform (CCP)? Let's have an example.

There is a third party service that we are interested in ingesting logs into Sentinel. The service exposes a public REST API endpoint. You can think of developing a Function App in Python, which runs once every 20 minutes and which connects to the REST API endpoint retrieving the data (*i*) and then ingesting it into Sentinel by calling the Log Analytics API (*ii*).

Three pain points:

- 1. You should have experience in coding which could be minimized by using a Logic App but not eliminated
- 2. You need to maintain the infrastructure (in this case the health of the Function App)
- 3. Cost associated with Function App executions

The Codeless Connector Platform resolves the issues mentioned above.

Key benefits include (from The Codeless Connector Platform - Microsoft Community Hub):

- Avoid writing lines of code to connect with publicly exposed REST APIs
- Scalable built in Poller as a service
- Configurable UI components for your connector
- Ingest Cost benefits
- Monitor your connectors: CCP integrates with Sentinel Connector Health message using which you can troubleshoot and get health messages.

This guide focuses on the legacy version of CCP – the new version is in preview, out of the scope here. Everything is based on the creation of a json file that defines the connector interface and the endpoint from which to retrieve the data. Nothing more. It uses the Sentinel backbone to perform Endpoint API calls.

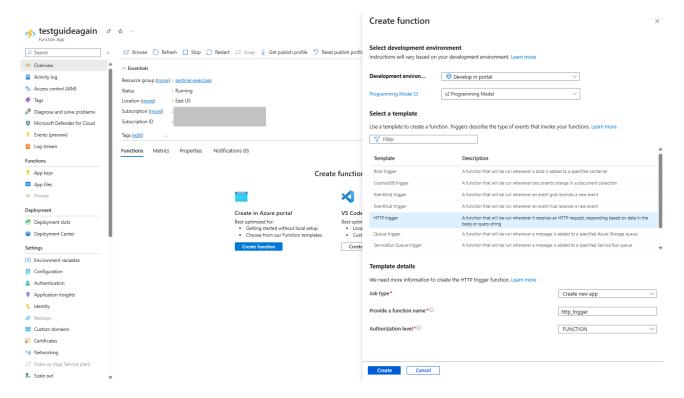
ARCHITECTURE

There is a third party service that we are interested in ingesting logs into Sentinel. The service exposes a public POST REST API endpoint. The public endpoint returns a json response as which has the following structure:

We want to store *name* and *surname* in the **TestData_CL** custom table.

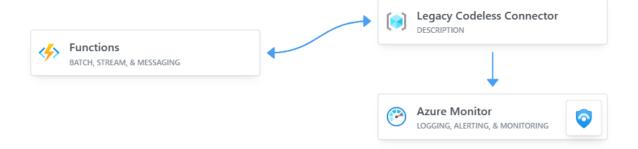
To simulate the third-party service you can create a Function App – for simplicity in Python. Some useful take-note during the creation process:

- Deploy Code and not Container Image
- Runtime stack Python Version 3.11
- Hosting options and plans: Consumption
- Enable Public Access
- Template HTTP trigger
- Job type create new app
- Authorization level FUNCTION



The python function is as follows (also available here)

```
import azure.functions as func
import logging
import json
import random
import string
app = func.FunctionApp(http_auth_level=func.AuthLevel.FUNCTION)
@app.route(route="http_trigger")
def http_trigger(req: func.HttpRequest) -> func.HttpResponse:
    logging.info('Python HTTP trigger function processed a request.')
    name = ''.join(random.choices(string.ascii_lowercase, k=5))
    surname = ''.join(random.choices(string.ascii_lowercase, k=5))
    return func.HttpResponse(
            #body=json.dumps([{"name": name, "surname": surname}]),
            body=json.dumps({
                "status": "OK",
                "items" : [{"name": name, "surname": surname}]
                }
            ),
            status code=200,
            mimetype="application/json"
```



CONNECTOR

In this section we analyse the structure of the ARM template used to deploy the connector. This is general structure

The two most important properties of the Json file are *connectorUiConfig* and *pollingConfig*: the first section contains the graphical interface of the connector (title, description, publisher, query examples, step-by-step description for deployment) and the second contains information on the how to recover data (endpoint, authentication, how often to make the API call, how long to wait in case of error).

The entire Json file is as follows (also available here)

```
"$schema": "https://schema.management.azure.com/schemas/2019-04-
01/deploymentTemplate.json#",
  "contentVersion": "1.0.0.0",
  "parameters": {
    "workspace": {
      "type": "string",
"defaultValue": ""
  "resources": [
"[concat('/subscriptions/',subscription().subscriptionId,'/resourceGroups/',resourceG
roup().name,'/providers/Microsoft.OperationalInsights/workspaces/',parameters('worksp
ace'),'/providers/Microsoft.SecurityInsights/dataConnectors/',guid(subscription().sub
scriptionId))]",
      "name":
"[concat(parameters('workspace'),'/Microsoft.SecurityInsights/',guid(subscription().s
ubscriptionId))]",
      "apiVersion": "2021-03-01-preview",
      "type": "Microsoft.OperationalInsights/workspaces/providers/dataConnectors",
      "kind": "APIPolling",
      "properties": {
        "connectorUiConfig": {
```

```
"title": "Personal Legacy CodelessConnector",
          "id":"LegacyCodelessConnector",
"publisher": "Mario Cuomo Dev",
          "descriptionMarkdown": "This connector is used to retrieve data from an
Azure Function <insert your API Endpoint>. Fetch every minute.",
          "graphQueriesTableName": "TestData_CL",
           "graphQueries": [
            {
               "metricName": "Total data received",
               "legend": "Audit Events",
               "baseQuery": "{{graphQueriesTableName}}"
            }
          ],
           "sampleQueries": [
               "description": "Distinct name value",
               "query": "{{graphQueriesTableName}}\n | distinct name\n"
           "dataTypes": [
               "name": "{{graphQueriesTableName}}",
               "lastDataReceivedQuery": "{{graphQueriesTableName}}\n
summarize Time = max(TimeGenerated)\n
                                                    | where isnotempty(Time)"
          ],
"connectivityCriteria": [
               "type": "SentinelKindsV2",
               "value": [
                 "APIPolling"
            }
          ],
           "availability": {
             "status": 1,
             "isPreview": false
           'permissions": {
             "resourceProvider": [
                 "provider": "Microsoft.OperationalInsights/workspaces",
                 "permissionsDisplayText": "read and write permissions are required.",
                 "providerDisplayName": "Workspace",
                 "scope": "Workspace",
                 "requiredPermissions": {
                   "action": true,
                   "write": true,
                   "read": true,
                   "delete": true
              }
            1
          },
"instructionSteps": [
               "title": "Authenticate against the Azure Function",
               "description": "Provide the Function API Key.",
               "instructions": [
```

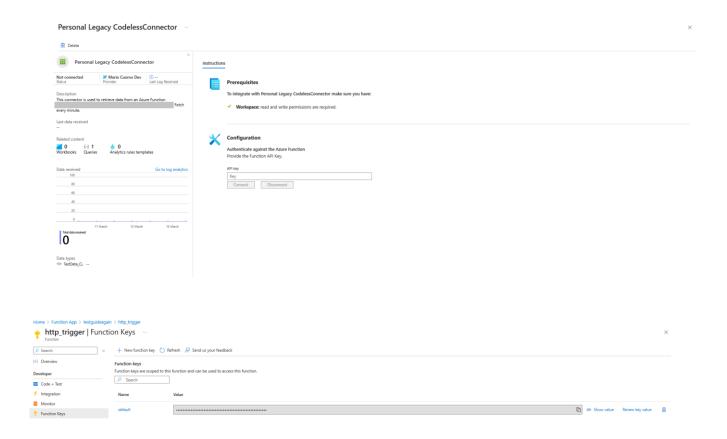
```
"type": "APIKey"
              ]
            }
          ]
        "pollingConfig": {
          "auth": {
            "authType": "APIKey",
            "APIKeyName": "x-functions-key",
            "IsAPIKeyInPostPayload": false
          },
          "request": {
            "apiEndpoint": "<insert your API Endpoint>",
            "rateLimitQPS": 2,
            "httpMethod": "Post",
            "queryTimeFormat": "yyyy-MM-ddTHH:mm:ssZ",
            "retryCount": 3,
            "queryWindowInMin": 1,
            "timeoutInSeconds": 20
          },
          "response": {
            "eventsJsonPaths": [
              "$.items"
            "format": "json",
            "successStatusJsonPath": "$.status",
            "successStatusValue": "OK"
        }
     }
    }
  ]
}
```

Some useful take-note during the creation process

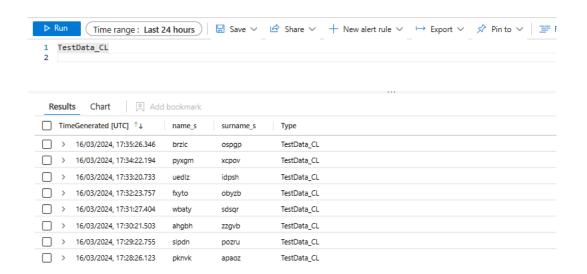
- L'apiEndpoint is like https://<FunctionAppName.azurewebsites.net/api/FunctionName
- Authentication is performed by passing the apikey for the x-functions-key in the header
- The data to be stored in the *TestData_CL* table (graphQueriesTableName property) is in the nested *items* object
- The connector is active by checking the *status* property of the object returned from the apiEndpoint

Once the connector is deployed, you can see it in Sentinel.

To connect, you need to provide the value of an available api key - even the default one.



Wait 30 minutes and you can start seeing data in Sentinel!



REFERENCE

Some useful links

- Codeless Connector Platform, techcommunity blog
 The Codeless Connector Platform Microsoft Community Hub
- Legacy Connector Platform Docs
 Legacy codeless connector for Microsoft Sentinel | Microsoft Learn
- LastPass Connector an example available in Azure-Sentinel official Github Repository

 <u>Azure-Sentinel/Solutions/LastPass/Data Connectors/LastPassAPIConnector.json at master · Azure/Azure-Sentinel · GitHub</u>
- Github Repository
 <u>mariocuomo/LegacyCodelessConnectorSentinel: Proof of concept for a Sentinel connector based</u>
 on the Codeless Connector Platform (github.com)
- Step-by-Step Youtube video https://youtu.be/g757IDY-8h4